

66/6T/LD

A



ASSISTANT COMMISSIONER OF PATENTS
Washington, DC 20231

DOCKET NUMBER: UK998093

Sir:

Transmitted herewith for filing is the Patent Application of:

Inventor(s): E K Kolodner & M J Trotter

For: VIRTUAL MACHINE MEMORY MANAGEMENT

Enclosed are:

- ☒ Patent Specification and Declaration
- ☒ 6 sheets of drawing(s)
- ☒ An assignment of the invention to International Business Machines Corporation (includes Recordation Form Cover Sheet)
- ☒ A certified copy of a United Kingdom application filed on 23 December 1998, Serial Number 9828334.4
- ☐ Information Disclosure Statement, PTO 1449 and copies of references

515 U.S. PTO
09/356532
07/19/99

The fee has been calculated as shown below:

For	Number Filed	Number Extra	Rate	Fee \$
Basic Fee				\$ 760.00
Total Claims	26 - 20	6	x 18 =	108.00
Independent Claims	3 - 3		x 78 =	
MULTIPLE DEPENDENT CLAIM PRESENTED			260 =	
TOTAL				<u>\$868.00</u>

☒ Please charge my Deposit Account No. 09-0468 in the amount of \$868.00. A duplicate copy of this sheet is enclosed.

☒ The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account 09-0468. A duplicate copy of this sheet is enclosed.

☒ Any additional filing fees required under 37 CFR §1.16.

☒ Any patent application processing fees under 37 CFR §1.17.

CERTIFICATE OF MAILING

I hereby certify that the above paper/fee is being deposited with the United States Postal Service as first class mail in an envelope addressed to the Commissioner of Patents and Trademarks, Washington, DC 20231

Date of Deposit July 19, 1999

Person mailing paper/fee Wendy R. Petrovich

Signature _____

Respectfully submitted

Jay P. Sbrollini
Registration No. 36,266
IBM Corp, IP Law Dept,
T J Watson Research Center, PO Box 218,
Yorktown Heights, New York 10598.

APPLICATION
FOR
UNITED STATES LETTERS PATENT

INTERNATIONAL BUSINESS MACHINES CORPORATION

VIRTUAL MACHINE MEMORY MANAGEMENTFIELD OF INVENTION

This invention relates to memory management in a
5 multithreaded runtime environment.

BACKGROUND OF INVENTION

The Java programming language has its origins in a
project undertaken by Sun Microsystems to develop a
10 robust programming environment that would meet the
technical challenges of the consumer device software
environment. The original consumer device projects were
eventually abandoned but the Java programming language
found itself being used on the World Wide Web to enable
15 cross platform operation of programs downloaded from the
internet. It is simple to use having similar features to
C++ such as the basic object orientated technology but
without some of the more complex features.

Typically, Java applications (source code) are
20 compiled by the Javac compiler into Java byte code
(intermediary code or pseudo object code) which can be
loaded and executed by a Java Virtual Machine (JVM) (see
Figure 1). The JVM provides an instruction set, memory
management and program loading capability that is
25 independent of the hardware platform on which it has been
implemented. The Java application source code is
compiled into architecture independent byte code and the
byte code is interpreted by a JVM on the target platform.
Java is designed to be portable and follows some defined
30 portability standards, which intend the source code to be
"write once, run anywhere". The Java byte code may be

further compiled into machine code (object code) for the target platform at which point the architectural independent nature of Java is lost.

5 The JVM is a software computing machine, effectively it simulates a hardware machine that processes Java byte code. The byte code is interpreted and processed by a JVM such as an Windows JVM running on a Intel personal computer platform. The JVM includes components for loading class files, interpreting the byte code, garbage
10 collecting redundant objects, and for managing multiple processing threads. The JVM may also include a Just-In-Time compiler to transform some or all the byte code into native machine code.

5 Multithreading is a feature built into the Java language to allow users to improve interactive performance by allowing operations to be performed while continuing to process user actions. Multithreading is similar to multitasking, but whereas multitasking allows many applications to run on the same system in several
20 processes, multithreading allows many routines (threads) in one application to potentially run in parallel within one process.

25 Garbage collection is the term used for describing how program objects are automatically discarded by the system after they are no longer useful.

For further information on garbage collection see Chapter 1 of 'Garbage Collection' by H Jones & R Lins, Wiley. Chapter 4 deals with Mark & Sweep techniques.

30 Many current implementations of Java use the classic mark-sweep-compact method of garbage collection as delivered in the base SUN JVM. References to the objects

that are being processed at any instant by the system are stored in the registers, one or more thread stacks and some global variables. The totality of objects that are maybe needed by the system can be found by tracing through the objects directly referenced in the registers, stacks, and global variables and then tracing through these "root" objects for further references. The objects in use by a system thereby form a graph rooted at the root objects and any extraneous objects are not part of this graph. Once all the objects in the graph are found, the remaining objects in the heaps may be discarded (garbage collected).

The traditional mark and sweep garbage collection method is described below in terms of pseudo code with respect to a single heap:

- Stop all threads causing the active registers for each thread to be stored in its stack
- Trace all stacks for object references - the local roots
- Trace all global variables object references - the global roots
- Trace through root set for references until no new object references (the sum of the local and global roots is the root set).
- Delete all objects in the single heap that are not referenced

Note that in a particular Java virtual machine the term 'global variables' may include variables in classes, the global JNI table, objects awaiting finalisation and strings which have been 'interned'.

There are problems with this technique in a multi-threaded and long running environment. The first problem is that in order to garbage collect all the threads must be stopped in order to work out what objects are unreachable (i.e., there are no pointers to them in the global or local variables and no pointers to them in other reachable objects). Various authors have attempted to solve this problem. One approach is an on-the-fly collector, which does not stop all threads, however, it cannot compact the reachable objects leading to fragmentation. Another approach are the generational scavenging schemes, which is to reduce the size of the set of traced objects by concentrating effort on the most recently allocated objects; however, these schemes must stop all of the threads. In an ideal world we would like to achieve a collector which works independently on all threads and compacts the local heap of the threads to maximise the free space available.

Another solution attempts to achieve this in a language (ML) other than Java by taking advantage of immutable objects which can be placed in thread-local heaps. An immutable object is non-modifiable and when such an object become reachable globally a copy of the object can be made in the global heap. Clearly this technique is only applicable to languages defining immutable objects.

Another approach moves an object into the global heap on first use. The difficulty here is that in order to move the object, references from elsewhere to it must be updated; in an environment where objects are referenced by handles this is made easier although there

are still cases where objects cannot be moved.
Unfortunately handles bring their own problems and the
IBM ports of the JVM have removed handles to improve
performance and remove the need to subdivide the heap
into handles and object spaces.

SUMMARY OF INVENTION

According to one aspect of the invention there is
provided a method of managing memory in a multi-threaded
processing environment including respective local thread
stacks and heaps and a global heap, said method
comprising:

creating an object in a thread heap; and monitoring
whether the object is reachable from anywhere other than
the thread stack or registers.

Preferably the method further comprises: associating
a local status with the object; and changing the status
of the object to global under certain conditions.

More preferably the method further comprises
deleting from the thread heap one or more local objects
when they are not reachable from a local root.

Advantageously where reachability is determined by
tracing from the local root.

More advantageously the status of an object in the
thread heap is changed to global if the object is
assigned to a static variable or if the object is
assigned to a field in a global object.

Hence this proposal concentrates on the case of
implementing a thread local heap which has not been done
before. In essence the solution is to keep track of
object references using write barriers placed in any

operation which assigns references eg putfield, putstatic and astore.

The putfield operation for example is a java bytecode which causes the value on the top of the stack to be placed in the object referenced below it on the stack. The object field to be updated is defined by the constant pool reference (2 bytes) which follows the bytecode. See p.325 'Java Virtual Machine Specification', Lindholm & Yellin 1997.

A flag is associated with each object in the heap. In practice the flag is easily provided by ensuring that all objects on the heap are allocated on an 8 byte boundary ensuring that the low three bits are 0. Two of these bits are currently used to denote pinned or free objects leaving a spare flag which we use to denote 'global'. When an object is allocated it is generally placed in a piece of storage associated with the allocating thread; the thread local heap. The global flag is unset at this point. From now on the write barriers implement the following rules:

- 1) if an object reference is assigned to a static (global) variable the object becomes global and all objects reachable from it also become global; and
- 2) if a reference to a non-global object is assigned to another object which is global then the referenced object and all objects that are reachable from it also becomes global.

Thus at any point the thread local heap will contain objects which have the global flag unset and some which have the flag set. Any objects with the flag unset can be garbage collected by the thread itself. The objects

marked global must at this stage be treated as reachable, pinned objects; they must not be moved or deleted. Any garbage collection policy can be adopted including compaction for the objects in the local heap, the only constraint is that we must of course compact into the potentially fragmented spaces left between the global objects. Note that the write barriers described above are extremely simple and can be implemented to have a very small effect on performance.

At some point the global objects will be moved from the local heaps into the global heap and this must be done using the current collection approach of stopping all threads. The belief is of course that the frequency of such events can be dramatically reduced and the work to be done also reduced. The global objects need to be removed and put in their own heap, once that is done the individual threads can be left to continue any tidying they need to do on their own heaps. The great improvement is that the moving of global objects is the only part which must be done whilst all program threads are stopped. Tidying of the local heaps can be then done in parallel. The current approach forces the compaction of the entire heap (moving objects) to be done by one thread and all others must wait for completion.

BRIEF DESCRIPTION OF DRAWINGS

In order to promote a fuller understanding of this and other aspects of the present invention, an embodiment will now be described, by way of example only, with reference to the accompanying drawings in which:

Figure 1 is a schematic representation of a platform supporting the embodiment of the present invention;

Figure 2 is a schematic representation of a Java Virtual Machine embodying the invention;

5 Figure 3 shows the structure of objects in a thread heap;

Figure 4 is a flow diagram depicting the memory management process of the invention;

10 Figure 5 is a flow diagram of the process of the write barrier; and

Figure 6 is a schematic representation of objects in a thread heap and the global heap of the JVM.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENT

15 Referring to Figure 1 there is shown a computer platform 10 such as pentium processor based PC typically comprising motherboard with processor, 32Mbytes RAM memory, and interface circuitry; 4G Byte hard drive; 20x CD ROM; SVGA monitor; keyboard and mouse. On power on the
20 platform loads into the memory an operating system 12 such as Windows NT v4 and subsequently a Java Virtual Machine (JVM) based on the Java Development Kit (JDK) v1.1 or v1.2. The Java virtual machine is capable of loading a Java application 16 from disk into memory so
25 that the application may be executed on the platform. The Java application 16 is object orientated program code and creates Java language objects 18A, 18B, 18C, 18D. An example of how the objects are built by the JVM is described with reference to Figure 3.
30

The JVM 14 comprises code components for the multitude of tasks it has to perform (see Figure 2). The components of relevance to the present embodiment comprise memory manager 20, Java interpreter 22 and just-in-time compiler 24. In operation the JVM creates memory space, the runtime data area 26, for storing the objects 18 and data for the application 16.

The runtime data area 26 comprises thread memory space 28A, 28B, 28n, global heap 34, class area 36 and compiled method area 38. The thread memory space 28 ideally stores local thread objects, local thread execution data and also objects marked global. Local objects are objects which are used by one thread alone and are not common between. Thread objects are stored in thread heaps 32A, 32B, 32n and thread execution data including object references and register data are stored in the thread stacks 30A, 30B, 30n. The global heap 34 stores objects which are common to more than one thread, that is global objects. A local object which is loaded into a thread heap 32A for processing by thread 28A may become global if it is later used by another thread, say thread 28B, and may remain in the thread heap 28A or be moved to the global heap 34 by 'stop everything'. Class area 36 stores the classes as they are loaded into the JVM, classes are defined as global because they are common to all the threads hence any field of a class is globally reachable. An object, which is an instance of a class, is placed in a thread heap and is initially local because it may only be used by that thread. An exception is instance objects of class Thread or Runnable which are global at creation because they can be reached by other

threads. Whereas an object referenced by a variable in a class is global because the object is reachable from the class which is reachable by all threads. The compiled method area 38 is used by the JIT compiler 24 to store native executable code compiled from the Java byte code.

The memory manager 20 includes an object allocator 21A and a garbage collector 21B which between them control the creation and deletion of objects within the thread heaps 32A, 32B, 32n and the global heap 34.

The Java interpreter 22 includes functionality to process the multitude of Java byte codes which make up the Java byte code function set. A write operation component 23A processes java write operations such as 'putfield', 'putstatic', 'aastore' using the operating system and microprocessor of the platform. Putfield and aastore are write operations which among other things assign a source object reference to a field in a target object. This means that the target object will contain a reference (or pointer) to the source object; the target object is the object that will be updated by the putfield operation. Putstatic assigns an object reference to a static variable in a class. A write barrier component 23B checks the status of the target object and assigns a local or global status to the source object depending on certain rules:

- 1) if an object reference (of a source object) is assigned to a static variable of a class the source object becomes global and all objects reachable from it also become global; and

2) if a non-global source object is assigned to a reference in a target object which is global then the referenced source object and all objects that are reachable from it become global.

5 The Java write operation 'putfield' may set a field within a target object to point at a source object using the operating system commands or platform operations . A 'putfield' write operation 23A together with the write barrier 23B is described below in pseudo platform code with comments.

10 If field to be updated in target object is of 'reference' type.

if (target is global) {Check to see to the global flag is set for the target

15 if (source is not global) {If so check to see if the source has already been set global

20 set source to global {If not already set then set source to global

25 set objects reachable by source to global {and set all objects reachable, (i.e., directly referenced by source object or referenced from another object reachable from the source object by the source object to global

30

```
set slot (target, field number, source)
                                {sets a field variable
                                within a target object to
                                point at a source object.
```

5

The last instruction code is the write operation 23A whereas the instructions before that constitute the write barrier 23B. Similar pseudo code may be provided for the putstatic and astore java commands, and other JVM actions, which store references in objects.

10

The JIT compiler 24 is similar to the Java interpreter 22 in that it contains a write operation component 25A and a write barrier component 25A. However instead of interpreting java byte code in the methods of objects and processing the operations in real time, native code similar to the pseudo code above is created and stored in the compiled method area 38 for execution directly by the platform and operating system whenever that particular method is invoked.

5

20

Objects 18 are stored in the thread heaps 32 using multiples of 8 bytes (see Figure 3). Each object has a length word attached which identifies the length of the object in bytes and allows objects in the heaps to be scanned sequentially from the start of the heap. The length word (4 bytes long in the embodiment) has three spare bits available because of the 8 byte alignment boundary and one of these bits is used as a flag for the object to store the local or global status. In the embodiment shown in Figure 3 each of the objects is 40 bytes long which is 5*8 bytes or '00101000' bytes in binary; the last three '000' being spare. This is

25

30

indicated at the high end of the word in each case. The low end of the word is spare and thus the first bit of the length word is used to indicate the local/global status, in this case object 18A and 18B are set to '1' to indicate that they are global and 18C is set to '0' to indicate that it is local.

The process of memory management is described with reference to Figure 4. Step 4.1 an object is created from a class stored in the class area 36. Step 4.2 check to see if the class is a 'global' class, i.e., a class all of whose instances are global or one whose instances we expect to quickly become global, whereby the object is assigned global and/or placed in the global heap (Step 4.7). Step 4.3 the size of the object is calculated to see whether it will fit in the thread heap. Step 4.4 if there is not enough space then garbage collection is performed to free up memory. Step 4.5 place the object in the thread heap memory along with the object length in multiples of eight. Step 4.6 Use a spare bit in the length attribute as a flag and set as local ('0'). The process ends at step 4.8.

The write operation and write barrier process are described with reference to Figure 5. Step 5.1 a write operation is called and intercepted by the write barrier which is integrated with the write operation code. Step 5.2 Check whether the object is being assigned to a static variable of a class and if so set the status of the object as global (step 5.4) and set all objects reachable by the object as global (step 5.5) before moving to step 5.6, if not so then do step 5.3. Step 5.3 Check whether the write operation assigns the object to

another global object, if so set the status of the object as global (step 5.4) and set all objects reachable by the object as global (step 5.5). Once complete continue with the write operation.

5 The garbage collector 21A for single thread garbage collection is described in terms of pseudo code with comments.

Trace stack in thread {Identify objects in use by
the thread - these are held
10 in the thread stack

Trace reference objects {Identify objects
referenced directly and
indirectly by the stack
15 objects.

Delete unused objects {those objects in the heap
which are not identified in
trace or are not identified
as global

Check heap size {If not enough memory then
increase size of the local
heap. If not desirable or
possible then do 'full'
20 garbage collection

25 Full garbage collection is traditional Mark/Sweep performed with all other threads stopped. During full garbage collection global objects are moved from local to the global heap if possible. Object movement is not
30 possible if conservative tracing is performed and the

object is referenced directly from the stack because of the uncertainty of the tracing.

The Java language pseudo code program below (together with comments) creates objects representing a bike with two wheels and a bell in a local thread which is put into a global shed. The bike and wheels are make global whereas the bell remains local. If a further new bike object were to be created and the thread heap had no more memory then the garbage collector would mark and sweep the bell object so long as it was not pointed to by the stack.

```

    // Demonstrate the Thread local heap in action.
    // For simplicity all niceties such as constructors
    have been omitted and the example
    // made deliberately unreal to illustrate the basic
    point. Objects generally start life
    // local and become global by assignment.
    // This is a fake class just to illustrate a static
    variable in operation.
class Shed { // This is a fake class just to illustrate
    a static variable in operation.

    static Bike;

    // The main method will run the sample code to
    illustrate creation of local
    // objects which later will become global.
    public static void main(String args[]) {
        Bike b = new Bike(); // build a new Bike object
        // The Bike class is loaded and
        initialised and an instance of Bike is created in the
        local heap. It is marked 'local'.

```

Bell c = new Bell(); // build a new Bell object.
The Bell class is loaded and initialised and an instance
of Bell is created in the local heap. It is marked
'local'.

5 Wheel w = new Wheel(); // build a new wheel
object. The Wheel class is loaded and initialised and an
instance of Wheel is created in the local heap. It is
marked 'local'.

10 b.wheel1 = w; // Assign the first wheel of the bike
Java bytecode putfield, write barrier detects that target
(b) is local and hence w remains local.

Shed.bike = b; // Assign b to the static field
in bike Shed. Java bytecode putstatic is used and thus b
is marked global. Since w is reachable from b then w is
also marked global.

w = new Wheel(); // build another new wheel
object. Another instance of Wheel is created in the
local heap. It is marked 'local'.

20 b.wheel2 = w; // Assign the second wheel of the
bike

// Java bytecode putfield, write
barrier detects that target (b) is global and hence w is
marked global also.

25 // Note that at this point we have built a bike, two
wheels and a bell. The bell is the only object which is
still local since the bike has been assigned to the
static variable 'bike'.

}
}

30 class Bike {
Wheel wheel1;

```
Wheel wheel2;
Bell bell1;
String name;
}
5 class Wheel {
  int diameter;
}
class Bell {
10 int volume;
}
```

In summary there is described a method and system for memory management in a virtual machine or operating system and in particular an object creation and garbage collection method and system. There is described a method and system of managing memory in a multi-threaded processing environment such as a java virtual machine comprising: creating an object in a thread heap; associating a status with the object and setting the status as local; using write barriers to change the status to global if the object is assigned to a static variable or if the object is assigned to a field in a global object; changing the status of all objects referenced by that object to global; and performing garbage collection by deleting from the thread heap, when memory space in the thread heap is required, one or more local objects which can not be traced to the thread stack.

The platform in the embodiment does not need to be pentium based nor is restricted to the hardware stated. Any platform which is capable of supporting JDK v1.1 or v1.2 would be capable of supporting the virtual machine

of embodiment. Although the embodiment is described with reference to a Java Virtual Machine it is not necessarily so restricted to a virtual machine but may be used in any environment where storage is garbage collected to free up memory in that environment. For example, such an environment could be an multithreaded Lisp runtime or the runtime for some other multithreaded garbage collected language.

Although we have described in the preferred embodiment the local/global status as a field in the object it may be implemented in other ways for instance as a separate table.

Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation.

Now that the invention has been described by way of a preferred embodiment, various modifications and improvements will occur to those person skilled in the art. Therefore it should be understood that the preferred embodiment has been provided as an example and not as a limitation.

CLAIMS

1. A method of managing memory in a multi-threaded
processing environment including respective local thread
stacks and heaps and a global heap, said method
comprising:

creating an object in a thread heap; and
monitoring whether the object is reachable from
anywhere other than the thread stack.

2. A method as claimed in claim 1 further comprising:
associating a local status with the object;
changing the status of the object to global under
certain conditions.

3. A method as claimed in claim 2 further comprising
deleting from the thread heap one or more local objects
when they are not reachable from a local root.

4. A method as claimed in claim 3 where reachability is
determined by tracing from the local root.

5. A method as claimed in claim 4 wherein the status of
an object in the thread heap is changed to global if the
object is assigned to a static variable or if the object
is assigned to a field in a global object.

6. A method as claimed in claim 3 further comprising
intercepting assignment operations to an object in a
thread heap to access whether the object status should be
changed.

7. A method as claimed in claim 6 wherein the multithreaded processing environment is a virtual machine.

5 8. A method as claimed in claim 7 wherein the virtual machine comprises an interpreter and the write operation code in the interpreter is modified to perform the checking of assignment of the object.

10 9. A method as claimed in claim 8 wherein the virtual machine comprises a just in time compiler wherein native compiled write operation code includes native code to perform the checking of assignment of the object.

15 10. A method as claimed in claim 9 further comprising using spare capacity in the object header for the flag.

20 11. A method as claimed in claim 10 further comprising using multiples of 2 or more bytes in a thread heap to store the objects whereby there is at least one spare bit in the object length variable and using the at least one spare bit as the flag.

25 12. A method as claimed in claim 11 further comprising moving objects whose status is global from the thread heap to a global heap.

30 13. A method as claimed in claim 12 further comprising compacting the reachable local objects in a thread heap.

14. A method as claimed in claim 1 wherein certain objects are associated with a global status on creation.

5 15. A method as claimed in claim 14 where said certain objects include class objects.

10 16. A method as claimed in claim 14 further comprising the step of analysing whether an object is likely to be made global and associating such an object with a global status on creation.

17. A method as claimed in claim 16 further comprising allocating objects assigned as global on creation to the global heap.

18. A system for managing memory in a multi-threaded processing environment comprising:
respective local thread stacks and heaps;
a global heap;
20 means for creating an object in a thread heap; and
means for monitoring whether the object is reachable from outside the thread heap.

25 19. A system as claimed in claim 18 further comprising means for associating a local status with the object and means for changing the status of the object to global under certain conditions.

30 20. A system as claimed in claim 19 further comprising means for deleting from the thread heap one or more local objects when they are not reachable from a local root.

21. A system as claimed in claim 20 further comprising:
means for changing the status of an object in the
thread heap to global if the object is assigned to a
static variable or if the object is assigned to a field
in a global object.

22. A computer program product stored on a computer
readable storage medium for, when executed on a computer,
performing a method of managing memory in a
multi-threaded processing environment including
respective local thread stacks and heaps and a global
heap, said method comprising:

creating an object in a thread heap; and
monitoring whether the object is reachable from
outside the thread heap.

23. A method as claimed in claim 22 further comprising:
associating a local status with the object;
changing the status of the object to global under
certain conditions.

24. A method as claimed in claim 23 further comprising
deleting from the thread heap one or more local objects
when they are not reachable from a local root.

25. A method as claimed in claim 24 where reachability
is determined by tracing from the local root.

26. A method as claimed in claim 25 wherein the status of an object in the thread heap is changed to global if the object is assigned to a static variable or if the object is assigned to a field in a global object.

11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
22

VIRTUAL MACHINE MEMORY MANAGEMENT**ABSTRACT**

5 This invention relates to memory management in a
virtual machine or operating system and in particular to
object creation and garbage collection. There is
described a method and system of managing memory in a
multi-threaded processing environment such as a java
10 virtual machine comprising: creating an object in a
thread heap; associating a status with the object and
setting the status as local; using write barriers to
change the status to global if the object is assigned to
a static variable or if the object is assigned to a field
15 in a global object; changing the status of all objects
referenced by that object to global; and performing
garbage collection by deleting from the thread heap, when
memory space in the thread heap is required, one or more
local objects which can not be traced to the thread
20 stack.

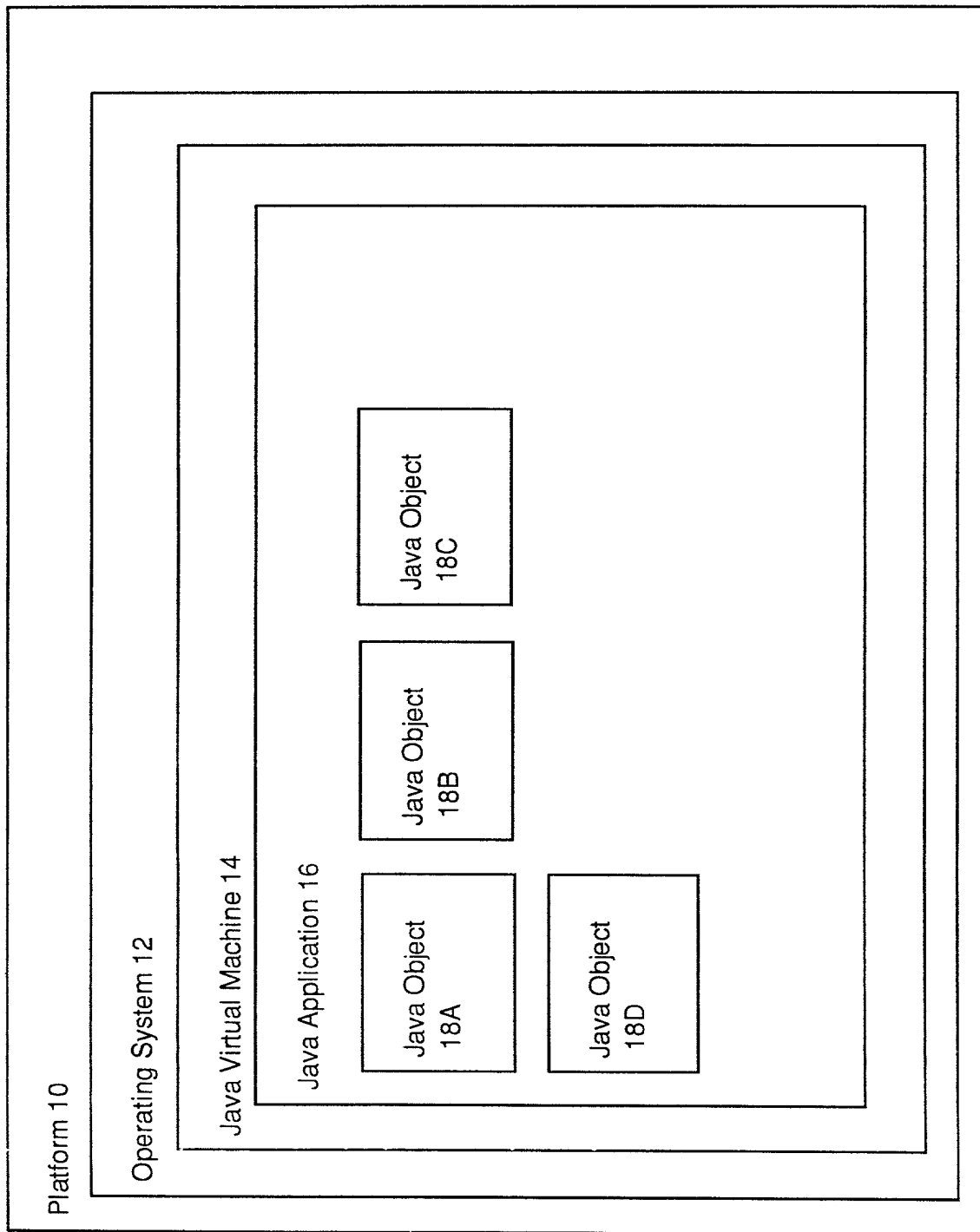


Figure 1

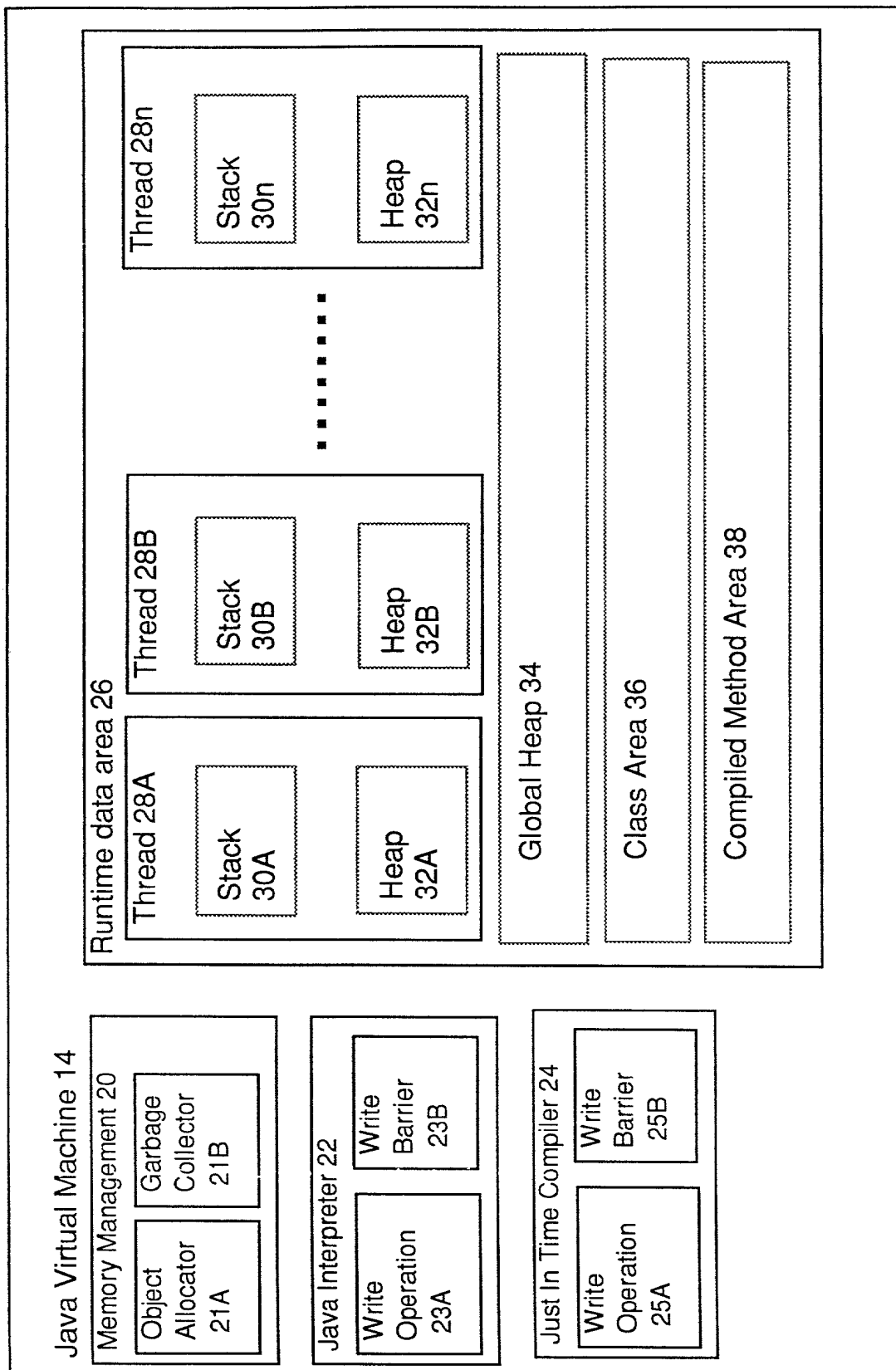


Figure 2

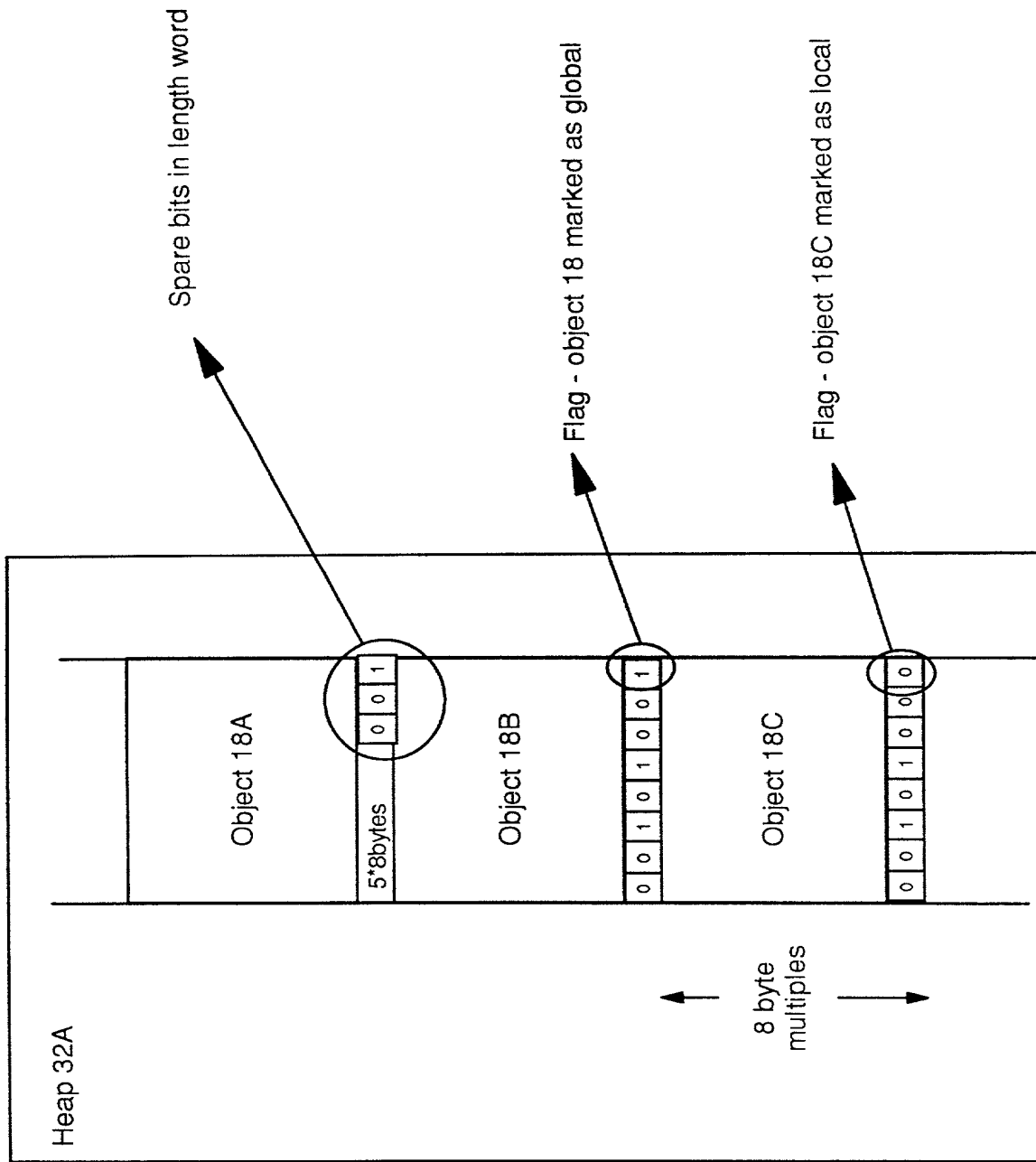


Figure 3

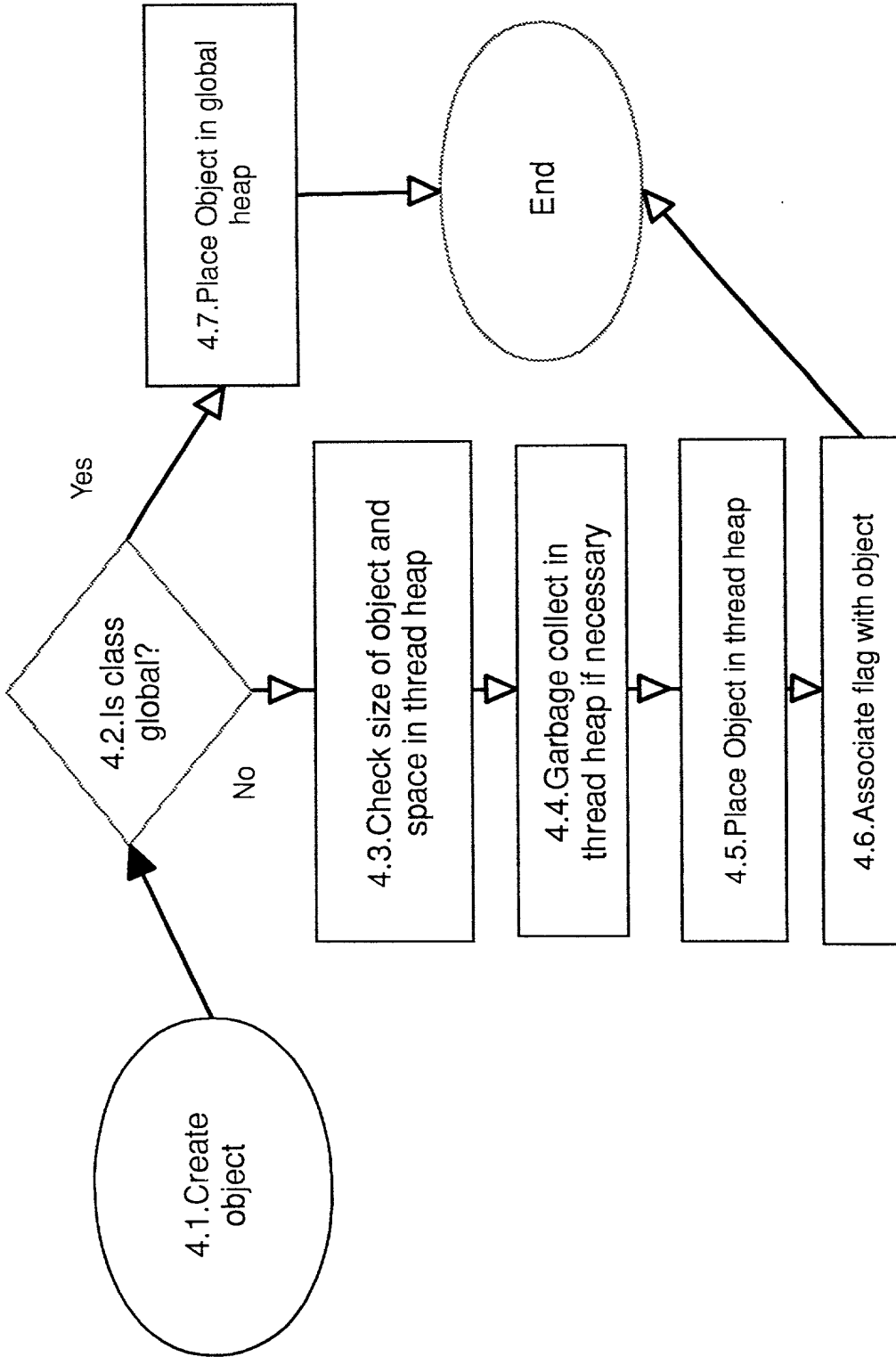


Figure 4

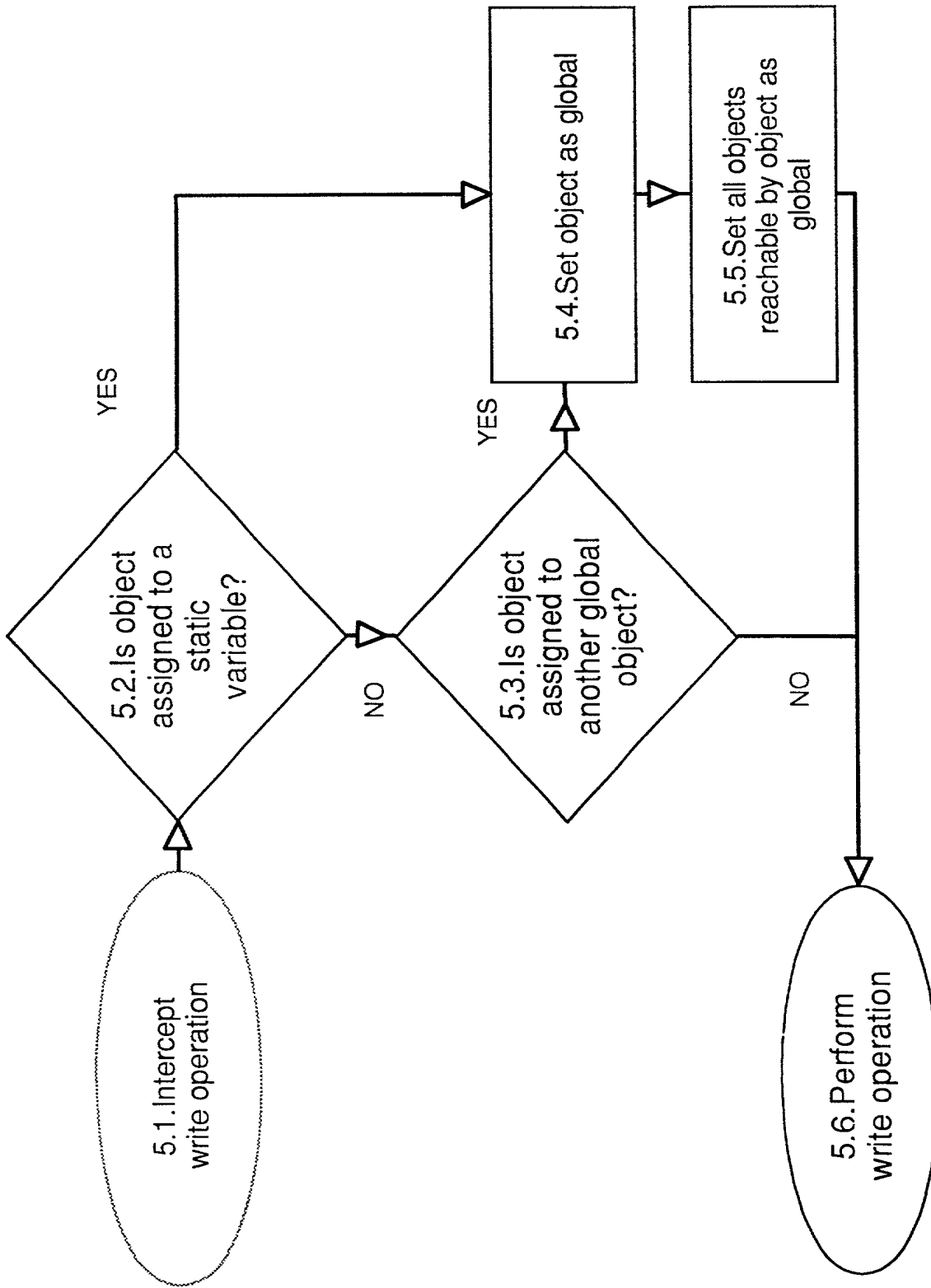


Figure 5

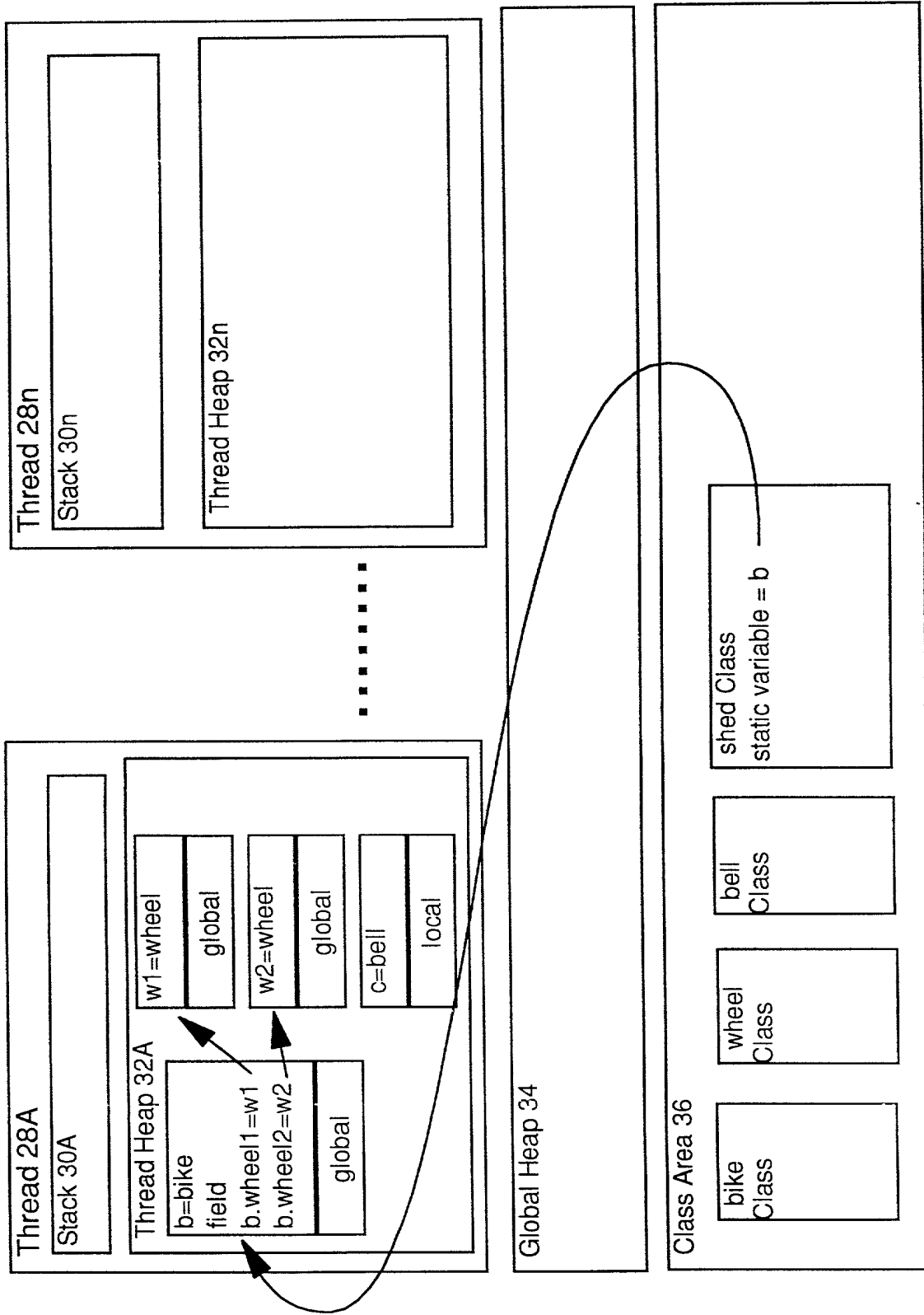


Figure 6

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name; I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

VIRTUAL MACHINE MEMORY MANAGEMENT

the specification of which (check one)

 X is attached hereto.

 was filed on
as Application Serial No.
and was amended on

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the patentability of this application in accordance with Title 37, CFR 1.56.

I hereby claim foreign priority benefits under Title 35, USC 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s):

9828334.4	United Kingdom	23 December 1998	<u> X </u>	Yes	<u> </u>	No
Number	Country	Filing Date Day/Month/Year		Priority Claimed		

I hereby claim the benefit under Title 35, USC 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, USC 112, I acknowledge the duty to disclose information material to the patentability of this application as defined in Title 37, CFR 1.56 which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

Prior U.S. Applications:

Serial No.	Filing Date	Status
------------	-------------	--------

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

POWER OF ATTORNEY: As a named inventor, I hereby appoint the following attorneys and/or agents to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

Manny W. Schechter, Reg. No. 31,722; William B. Porter, Reg. No. 33,135; Douglas W. Cameron, Reg. No. 31,596; Kevin M. Jordan, Reg. No. 40,277; Stephen C. Kaufman, Reg. No. 29,551; Richard M. Ludwin, Reg. No. 33,010; Daniel P. Morris, Reg. No. 32,053; Louis J. Percello, Reg. No. 33,206; Jay P. Sbrollini, Reg. No. 36,266; Stephen S. Strunck, Reg. No. 28,672; Robert P. Tassinari, Reg. No. 36,030; Robert M. Trepp, Reg. No. 25,933; Marc D. Schechter, Reg. No. 28,989; Louis P. Herzberg, Reg. No. 41,500.

Send all correspondence to:

Jay P. Sbrollini
IBM Corp, IP Law Dept,
T J Watson Research Center, PO Box 218,
Yorktown Heights, New York 10598.
Telephone: (914)945- 2587

C.K.K. Elliot
FULL NAME OF SOLE OR FIRST INVENTOR: ~~Elliot~~ Karl KOLODNER

INVENTOR'S SIGNATURE:

Elliot Karl Kolodner DATE: *April 29, 1999*

RESIDENCE:

40 Hanna Street, Haifa 2847, Israel

CITIZENSHIP:

Israel

POST OFFICE ADDRESS:

As above

FULL NAME OF SECOND INVENTOR: Martin John TROTTER

INVENTOR'S SIGNATURE:

Martin John Trotter DATE: *June 1st 1999*

RESIDENCE:

2 Broadgate Cottages, Potters Heron Close, Ampfield,
Romsey, Hampshire SO51 9BX, United Kingdom

CITIZENSHIP:

British

POST OFFICE ADDRESS:

As above

FULL NAME OF THIRD INVENTOR:

INVENTOR'S SIGNATURE:

DATE:

RESIDENCE:

CITIZENSHIP:

POST OFFICE ADDRESS: